KATHOLIEKE
UNIVERSITEIT
LEUVEN

DEPARTEMENT TOEGEPASTE ECONOMISCHE WETENSCHAPPEN

RESEARCH REPORT 0250

THE COMPLEXITY OF GENERATING ROBUST RESOURCE-CONSTRAINED BASELINE SCHEDULES

by

R. LEUS
W. HERROELEN

D/2002/2376/50

The Complexity of Generating Robust Resource-Constrained Baseline Schedules

Roel Leus and Willy Herroelen

November 2002

Operations Management Group
Department of Applied Economics
Katholieke Universiteit Leuven
Naamsestraat 69, B-3000 Leuven (Belgium)
Phones +32 16 32 69 67 and +32 16 32 69 70
Fax +32 16 32 67 32
e-mail: <first name>.<name>@econ.kuleuven.ac.be

The Complexity of Generating Robust Resource-Constrained Baseline Schedules

Roel Leus‡ and Willy Herroelen

Abstract

Robust scheduling aims at the construction of a schedule that is protected against uncertain events. A stable schedule is a robust schedule that will change little when variations in the input parameters arise. Robustness can also be achieved by making the schedule makespan insensitive to variability. In this paper, we describe models for the generation of stable and insensitive baseline schedules for resource-constrained scheduling problems and present results on their complexity status. We start from a project scheduling viewpoint and derive results on machine scheduling sub-problems.

Keywords: scheduling; uncertainty; robustness; stability; sensitivity; complexity

‡Research assistant of the Fund for Scientific Research, Flanders (Belgium) (F.W.O.)

1. Introduction

The project scheduling literature largely concentrates on the generation of a precedence and resource feasible schedule that 'optimizes' the scheduling objective(s) and that should serve as a *baseline schedule* or *pre-schedule* for executing the project (the two terms are used interchangeably throughout the paper). During project execution, however, project activities are subject to considerable uncertainty, which may lead to numerous schedule disruptions. *Proactive (robust) project scheduling* aims at the construction of a baseline schedule that takes into account uncertainty information (for instance information about the variability in activity durations) and that is protected against uncertain events that may occur during project execution. Several techniques for proactive scheduling have recently been published. The majority of publications are in the real-time shop scheduling literature, we refer to Davenport and Beck [6] for a review. Herroelen and Leus [11] give a recent survey of the existing approaches for dealing with uncertainty in project scheduling.

Stable scheduling or *solution robustness* refers to the particular case of robust scheduling where the objective is to minimize the anticipated deviation between the pre-schedule and the executed schedule. *Quality robustness* on the other hand is achieved when the objective function, typically the schedule makespan, is *insensitive* to variability. In view of achieving stability, various algorithms have been proposed that use a *match-up point*, described by Akturk and Gorgulu [2] as the time instance "where the state reached by the revised schedule is the same as the initial schedule," when action is undertaken after a machine breakdown. They continue, "the pre-schedule can be followed if no disruption occurs." Robust scheduling already builds protection into the pre-schedule, which makes it proactive rather than reactive.

The objective of this paper is to elaborate on work performed by Herroelen and Leus [10] on the generation of stables schedules without resource constraints, and by Leus and Herroelen [17] on robust resource allocation. We examine the complexity of resource allocation and schedule development with robustness objectives. We study the case of joint resource allocation and scheduling (Section 4) as well as the problem of stable resource allocation for a given input schedule (Section 3). As a preliminary, Section 2 introduces most of the notation and summarizes the polynomially solvable problem of stable schedule development in the absence of resource constraints. Although the approaches taken are fundamentally different, the research in this paper can be classified among existing

studies of complexity in robust scheduling, we refer the interested reader to Adiri et al. [1], Daniels and Carrillo [4], Daniels and Kouvelis [5] and Unal et al. [21].

2. Stability in the absence of resource constraints

Consider an activity-on-the-node project network $G(N,A)$, where N denotes the set of nodes (activities) and A is the set of arcs representing the finish-start zero-lag precedence relations. Node 0 is a dummy start activity and node $n=|N-1|$ a dummy end; these activities are predecessor and successor to all tasks that have no other predecessors or successors, respectively. We assume G to be acyclic and A to be minimal, excluding redundant precedence relations. Without loss of generality, we require $\forall(i,j)\in A$: $i<j$. TA designates the transitive closure of A: $(i,j)\in TA$ if there is a path from i to j in G.

It is assumed that all uncertainty during project execution can be captured by variability in activity durations. The stochastic variable representing the duration of activity $i\in N$ is denoted by D_i, the vector of stochastic variables by $\mathbf{D}$. A particular realization (sample) of $\mathbf{D}$ is written as $\mathbf{d}(d_0,\ldots,d_n)$. For a realization $\mathbf{d}$, a schedule S is defined by an $(n+1)$-vector of start times $\mathbf{s}(s_0,\ldots,s_n)$. Every $\mathbf{s}$ implies an $(n+1)$-vector of ending times $\mathbf{e}$, $s_i=e_i+d_i$; $\forall i\in N$. For a given schedule S and $(i,j)\in TA$, the *pairwise float* $F_{ij}(S)$ is defined as $s_j(S)-e_i(S)$. $F_{ij}(S)$ is undefined for $(i,j)\notin TA$.

For the development of a pre-schedule, we impose a project deadline ω and start with a set of deterministic baseline durations $\mathbf{d}$. We associate a probability of disruption p_i with every activity i (i=0,1,…,n–1), with $\sum_{i=0}^{n-1} p_i = 1$: we assume that *exactly one* disturbance will occur in the network, in the form of an increase in the duration of a single activity. This setting should not be seen as one where always exactly one disturbance will take place; the underlying idea is that disturbances are sufficiently sparse and spread over time and throughout the project network so that we can assume that the effect of one disturbance will not interact with the effects of another. The dummy end node has disruption probability p_n=0, while p_0 is the probability that the dummy start node, i.e. the entire project, starts later than initially anticipated. The random variable L_i denotes the increase in baseline duration d_i of activity i *if* it is disturbed. We assume all L_i to be discrete with probability mass function (pmf) $g_i(\cdot)$. This pmf associates nonzero probability with positive values $l_{ik}\in\Psi_i$, where Ψ_i denotes the set of disturbance scenarios for the duration of activity i; $\sum_{k\in\Psi_i} g_i(l_{ik})=1$. A non-negative cost c_i is incurred per unit time overrun on the start time of activity i; c_0=0. The *expected weighted deviation in start times in the realized schedule* from those in the *pre-schedule* is used as stability measure for a schedule S, or in other words, the objective function we wish to minimize is $\sum_{j=1}^{n} c_j(ES_j(S)-s_j(S))$,

with E the expectation operator and $S_j(\boldsymbol{S})$ a random variable representing the actually achieved starting time of activity j upon execution of the project. We impose that during project execution, activities cannot be started earlier than their planned starting times in the baseline schedule, i.e. $s_i(\boldsymbol{S}) \leq S_i(\boldsymbol{S})$, which guarantees that the baseline schedule is realized if all goes as planned (no disruptions). Makespan protection is the special case where $c_n>0$ and $c_i=0$, $i\neq n$.

In this section, we assume no resource constraints are present. Let $P(i,j)$ denote any path from i to j in $G(N,A)$. For a given schedule $\boldsymbol{S}$, define $MSPF_{ij}(\boldsymbol{S})=\min_{paths\ P(i,j)} \sum_{edges\ (q,r)\ in\ P} F_{qr}(\boldsymbol{S})$ as the minimum sum of pairwise floats of all edges on any path $P(i,j)$. $MSPF_{ij}(\boldsymbol{S})$ represents the protection of the activity start time $s_j(\boldsymbol{S})$ against a disruption of the duration of activity i. In the remainder of this paper, we mostly omit the argument indicating the schedule, as there is little danger of confusion. We can compute S_j as $\max\{s_j\ ;\ \max_{(i,j)\in A}\{S_i + D_i\}\}$, and so

$$ES_j = s_j + \sum_{i:(i,j)\in TA} p_i E(\max\{0; L_i - MSPF_{ij}\}).$$

Hence, we can write the objective function as

$$\begin{aligned}\min \sum_{j=1}^{n} c_j(ES_j - s_j) &= \min \sum_{(i,j)\in TA} p_i c_j E(\max\{0; L_i - MSPF_{ij}\}) \\ &= \min \sum_{(i,j)\in TA} p_i c_j \sum_{k\in\Psi_i} g_i(l_{ik}) \max\{0; l_{ik} - MSPF_{ij}\} \qquad (1)\end{aligned}$$

If the delay in the start time of activity j due to a duration increase l_{ik} of activity i according to disturbance scenario k is written as $\Delta_{ijk}=\max\{0\ ;\ l_{ik} - MSPF_{ij}\}$, we can obtain the following linear program (Herroelen and Leus [10]):

(EWD1) $\min \sum_{(i,j)\in TA} \sum_{k\in\Psi_i} p_i c_j g_i(l_{ik}) \Delta_{ijk}$ (2)

subject to

$s_i + d_i + \lambda_{ij} + MSPF_{ij} = s_j \qquad \forall (i,j)\in TA$ (3)

$s_n \leq \omega$ (4)

$l_{ik} - MSPF_{ij} \leq \Delta_{ijk} \qquad \forall (i,j)\in TA\ ,\ \forall k\in\Psi_i$ (5)

$\Delta_{ijk},\ s_i,\ MSPF_{ij} \geq 0$ (6)

Eqs. (3) compute the $MSPF$ values: λ_{ij} is the longest path of activity durations from activity i to j (excluding d_i and d_j). At the same time, the precedence constraints are imposed: for every $(i,j)\in A$, $s_i+d_i \leq s_j$. Eq. (4) imposes project deadline ω. Eqs. (5) allow us to linearize the objective (1). Eqs. (6) are the non-negativity constraints. The model can be simplified by eliminating variables $MSPF_{ij}$, and the resulting LP can be rearranged as the dual of a minimum cost network flow problem with $|N|$ nodes and $1+|A|+\sum_{(i,j)\in TA}|\Psi_i|$ arcs (Herroelen and Leus [10]). In conclusion, the robust scheduling problem without resource constraints can be solved in polynomial time.

3. Resource allocation for a given schedule

We present the concept of resource flow networks in Section 3.1. Section 3.2 studies the complexity of resource allocation for an input schedule.

3.1 Resource flow networks

When resource restrictions are introduced, a so-called *resource flow network* can be used to represent the flow of resources across the activities of the project network (cfr. Leus and Herroelen [17]). Assume that the execution of a project requires a set of renewable resource types R with constant availability a_k, $k=1,...,|R|$. Activity i requires $r_{ik} \leq a_k$ units of resource type k during each period of its execution. As an example, consider the project and associated feasible schedule shown in Figure 1. The project network is shown in Figure 1(a) in activity-on-the-node format. Activities 0 and 5 are dummies with zero duration and zero resource requirement. For each activity, the baseline duration and per period requirement for a single renewable resource type are shown above the corresponding node ($r_i:=r_{i1}$). The resource type is assumed to have a constant per period availability of 3 units. The feasible schedule shown in Figure 1(b) minimizes the project duration (the makespan equals the critical path lower bound).

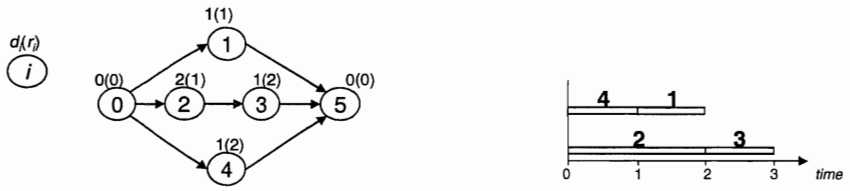

Figure 1. Example project and feasible schedule

Figure 2(a) represents a possible resource flow. We see that, for instance, the dummy start activity passes one unit of the resource to activity 2 and two units to activity 4, representing the dispatching of the resources into the project. The resource unit that executes activity 2 is transferred to activity 3 and on to dummy end activity 5, indicating that the resource does not contribute to the project after it completes its work on activity 3. A corresponding resource allocation is pictured in Figure 2(b), in which the 3 resource units are identified as M_1, M_2 and M_3. We notice that sometimes, more than one resource allocation can be associated with a given flow network. Nevertheless, with respect to the objective function, they are essentially

equal, and in the remainder of this paper, we use the terms 'resource allocation' and 'resource flow' interchangeably.

(a) (b)

Figure 2. Resource flow network and associated resource allocation

The technological precedence constraints of Figure 1(a) can be augmented with the extra 'resource links' $C(f)$ implied by the resource flow f in Figure 2(a) (dotted arcs), yielding a resource-unconstrained network. We define a resource flow f to be *compatible* with a schedule S if $\forall(i,j)\in TA\cup C(f)$: $e_i(S) \le s_j(S)$, or in other words $C(f) \subseteq \zeta(S) = \{(i,j)\in N\times N \mid (i,j)\notin TA \wedge e_i(S) \le s_j(S)\}$. The partial order resulting from $TA\cup C(f)$ defines an early start policy (Igelmund and Radermacher [12], Leus and Herroelen [17]), and all compatible schedules can be obtained with resource allocation f.

3.2 Resource allocation for an input schedule

The first basic problem to be treated in this paper is the following (with U a rational number):

ALLOCATION (ALL). *Can a feasible resource flow f be found for a given feasible schedule* S *such that* $\sum_{(i,j)\in TA\cup C(f)} \sum_{k\in\Psi_i} p_i c_j g_i(l_{ik}) \Delta_{ijk} \le U$ *and* $C(f) \subseteq \zeta(S)$?

ALL asks whether a resource allocation exists that is compatible with a feasible input schedule S and provides sufficient protection against uncertainty. The goal is to guarantee that S is realized if all goes as planned, so if no disruptions occur.

Decision problem ALL can be shown to be *NP*-complete. Motivated by the desire to find 'minimal' *NP*-complete sub-problems (Garey and Johnson [8]) we shall go about by proving that ALL is already *NP*-complete for the case of a single renewable resource ($|R|=1$) with availability $a=2$, both for makespan protection and for stability in job starting times. Remark that makespan protection is not implied by stability anymore (there is no dummy end job anymore), and that $a=1$ involves no allocation.

Consider a parallel machine schedule S in which n_k denotes the total number of (non-dummy) jobs assigned to machine k and let $[k,i]$ denote the job that is

scheduled in the i-th position on this machine. For a given due date $\omega \geq \max_{k=1,\ldots,a} e_{[k,n_k]}$, the total float of job $[k,i]$ can be written as

$$TF_{[k,i]} = \omega - e_{[k,i]} - \sum_{j=i+1}^{n_k} d_{[k,j]}. \tag{7}$$

For makespan protection, we set $\omega = \max_{k=1,\ldots,a} e_{[k,n_k]}$. This setting can be seen within the framework of ALL by assuming A to be empty except for pairs $(0,i)$ and (i,n), $i=1,\ldots,n-1$, $r_{i1}=1$ for each job $i\neq 0,n$, and $r_{01}=r_{n1}=p_0=s_0=0$. Additionally, $c_i=0$, $i\neq n$, and $c_n=1$, $s_n=\omega$ and $TF_{[k,i]}=MSPF_{[k,i]n}$. If we consider a single disruption scenario for each activity i with length $l_i:=l_{i1}$, the objective function can be written as $\sum_{i\in N\setminus\{n\}} p_i(l_i - TF_i)^+$, and we obtain decision problem 2MASEN:

2-MACHINE ALLOCATION SENSITIVITY (2MASEN). *Given the availability of two parallel machines, can a feasible machine allocation be found for a feasible input schedule* S *such that* $\sum_{k=1}^{2}\sum_{i=1}^{n_k} p_{[k,i]}(l_{[k,i]} - TF_{[k,i]})^+ \leq U$ *?*

For the particular case where each $(l_i - TF_i)$ is non-negative, which can be achieved by choosing each l_i sufficiently large (for instance $l_i \geq \omega$), minimization of the objective corresponds with maximization of

$$\sum_{i\in N\setminus\{0,n\}} p_i TF_i = \sum_{k=1}^{2}\sum_{i=1}^{n_k} p_{[k,i]} TF_{[k,i]} = \sum_{i\in N\setminus\{0,n\}} p_i(\omega - e_i) - \sum_{k=1}^{2}\sum_{i=1}^{n_k} p_{[k,i]} \sum_{j=i+1}^{n_k} d_{[k,j]}. \tag{8}$$

In other words, in the described setting, 2MASEN boils down to verifying whether there exists a machine allocation such that

$$\sum_{k=1}^{2}\sum_{i=1}^{n_k} p_{[k,i]} \sum_{j=i+1}^{n_k} d_{[k,j]} \leq U - \sum_{i\in N\setminus\{0,n\}} p_i(l_i - \omega + e_i). \tag{9}$$

Consider now the parallel machine scheduling problem with the objective of minimizing the sum of weighted completion times (problem $Pm||\sum w_jC_j$). The associated decision problem *DmS* has been shown to be (ordinarily) ***NP***-complete even for $m=2$ (Bruno et al. [3], by reduction from KNAPSACK). For the same set of jobs as 2MASEN, the expression to be minimized by $P2||\sum w_jC_j$ is

$$\sum_{i\in N\setminus\{0,n\}} w_i e_i = \sum_{k=1}^{2}\sum_{i=1}^{n_k} w_{[k,i]} \sum_{j=1}^{i} d_{[k,j]}, \tag{10}$$

where w_i denotes the weight of job i. Without loss of generality, we can subtract the constant term $\sum_{i\in N\setminus\{0,n\}} w_i d_i$ from (10). *D2S* boils down to answering the question whether for a given integer L there exists a schedule for $P2$ such that

$$\sum_{k=1}^{2}\sum_{i=1}^{n_k} w_{[k,i]} \sum_{j=1}^{i-1} d_{[k,j]} \leq L. \tag{11}$$

For a *given* selection of jobs to be processed on a particular machine, the *weighted shortest processing time first* (WSPT) rule, where the jobs are ordered on the machine in non-decreasing order of the ratio d_j/w_j, minimizes the weighted sum of completion

times (Pinedo [19]). In other words, it is the assignment of jobs to the machines, not the ordering of the assigned jobs on a machine, that makes the scheduling problem intractable. Thus, it can be seen that the bound in (11) can be achieved if and only if a schedule exists for which

Eq. (11) *holds and every machine has a WSPT order.* (12)

Solution of this decision problem reduces to solving an instance of 2MASEN with reversed time horizon, as follows. Order all jobs in *non-increasing* order of the ratio d_j/w_j, which can be done in $O(n\log n)$ time. Construct the input schedule S by putting all the jobs in this order in a contiguous chain, and assign probabilities $p_i:=w_i/\sum_j w_j$. A machine allocation satisfying (12) can be found if and only if the answer to 2MASEN is 'yes' when $U=L/\sum_j w_j+\sum_{i\in N\setminus\{0,n\}} p_i(l_i-\omega+e_i)$. In other words, *D2S* polynomially reduces to 2MASEN, which is therefore ordinarily *NP*-complete. As a corollary, we obtain that ALL is also ordinarily *NP*-complete. The proof can be extended to show strong *NP*-completeness for free number of machines.

An alternative and somewhat shorter proof is possible that directly reduces PARTITION to 2MASEN, by inserting two 'enforcer' jobs in parallel at the start of a schedule with all jobs (items) in series, both with disruption length equal to the sum of the durations (sizes) divided by 2, and asking whether a schedule exists with objective zero. Nevertheless, the proof was useful to introduce the notation and model, since it will again be referred to below (Section 4.1).

We define the following problem:

2-MACHINE ALLOCATION STABILITY (2MASTA). *Given the availability of two parallel machines, can a feasible machine allocation be found for a feasible input schedule* S *such that* $\sum_{k=1}^{2}\sum_{i=1}^{n_k} c_{[k,i]}\sum_{j=1}^{i-1} p_{[k,j]}(l_{[k,j]}-MSPF_{[k,i][k,j]})^+ \le U$ *?*

2MASEN reduces to 2MASTA, by inserting an enforcer job at the end of S as input to 2MASTA.

It is important to see that the evaluation of the objective function of ALL for a *given* solution (resource allocation) can be performed in polynomial time ($O(n^3+n^2\max_{i\in N}|\Psi_i|)$), in other words, the intractability of the PERT problem (Hagstrom [9]) that complicates the search for a general early start policy (Igelmund and Radermacher [12]; Möhring and Radermacher [18]; Stork [20]) is not an issue here. Rather, the resource allocation aspect itself seems to induce the complexity of the problem.

4. Joint resource allocation and scheduling

In the previous section we studied the resource allocation problem for an input schedule. In this section we analyze the complexity of various sub-problems of the joint resource allocation and scheduling problem, for a rational number U:

SCHEDULING AND ALLOCATION (SAA). *Given a project deadline ω, can a feasible resource flow f and compatible feasible schedule be found for which* $\sum_{(i,j)\in TA\cup C(f)}\sum_{k\in\Psi_i} p_i c_j g_i(l_{ik})\Delta_{ijk} \leq U$ *?*

Sub-problems are referred to without formal definition; they are constructed from SAA in a similar way as 2MASEN and 2MASTA from ALL. ALL reduces to SAA by adding for each activity i extra enforcer activities k_{1i} and k_{2i}, predecessor and successor of i, respectively, with zero resource usage and durations $d_{k_{1i}} = s_i(S)$ and $d_{k_{2i}} = \omega - e_i(S)$, such that each original activity i is 'fixed' at its starting time in S. As a result, SAA is at least *NP*-complete in the ordinary sense.

We ask the reader to consider the general deterministic resource-constrained project scheduling problem (RCPSP). Its decision problem version (checking for the existence of a feasible schedule subject to a deadline) is strongly *NP*-complete, which means that the extension to test whether a deadline *and* a bound on a variability performance measure can be respected, is also strongly *NP*-complete. We will pay attention to the specific case where the imposed scheduling deadline ω is not restrictive, meaning that ω is at least as large as the deterministic minimum makespan (or a heuristic solution) obtained when disregarding variability considerations. Even when the deadline is non-restrictive, ordinary *NP*-completeness of the decision version of $P2||C_{max}$ (reduction from PARTITION) implies ordinary *NP*-completeness of the 2-machine SAA with sensitivity objective, by giving the jobs a disruption length equal to ω minus the bound on the makespan (cfr. Section 4.2).

In the sequel, we investigate the basic single machine problem (Section 4.1), and we also study the complexity of SAA when the number of machines is a free parameter (Section 4.2), when precedence constraints are allowed (Section 4.3) and when different release dates are admitted (Section 4.4).

4.1 Single machine problem without precedence constraints or release dates

We will show that for the stability objective, even the 1-machine SAA is ordinarily *NP*-complete. For the sensitivity objective, the restrictive 1-machine problem is trivially solvable: all semi-active or 'left-justified' schedules are optimal.

We will show that $P2||\sum w_jC_j$ (cfr. Section 3.2) reduces to the single machine SAA with stability objective. For an instance of *D2S*, we have input parameters d_i, the durations of the jobs to be scheduled, and w_i, the weights of the jobs. *D2S* asks whether for a given integer L there exists a schedule for *P2* such that (11) holds.

We associate with an arbitrary instance of *D2S* an instance of single machine SAA, as follows. The jobs to be scheduled remain the same, however, we now choose durations $d_i'=1$ for each job. Each job has a single disruption scenario $l_{i1}=1$ and probability $p_i:=d_i/\sum_j d_j$, and we choose $\omega=n=(n-1)+1$ (=number of jobs plus one, so a feasible schedule always exists). The cost coefficients are selected as $c_i=w_i$, $i=1,\ldots,n-1$.

We first show that for a *given* order of the jobs, scheduling the available float of 1 time unit contiguously is always a dominant decision for SAA. This can be seen as follows: we have to divide the float of one time unit across the buffers $F_{[i][i+1]}$ ($i=0,\ldots,n-1$), with $F_{[0][1]}$ and $F_{[n-1][n]}$ the float before the first and after the last job, respectively. The objective function is $\sum_{j=1}^{n-2} p_{[j]} \sum_{i=j+1}^{n-1} c_{[i]} - \sum_{j=2}^{n-1} F_{[j-1][j]} (\sum_{i=1}^{j-1} p_{[i]})(\sum_{i=j}^{n-1} c_{[i]})$, and all $p_{[i]}$ and $c_{[i]}$ are constant for a fixed job order. Constraint $\sum_{i=0}^{n-1} F_{[i][i+1]}=1$ is imposed on the schedule by the choice for ω, such that it is never a dominated solution to assign all float (=1) to the buffer with highest coefficient in the second term of the objective expression. In conclusion, we restrict the search for an optimal schedule to solutions with a buffer of size 1 somewhere in the schedule, and given the disruption lengths of 1, this divides the jobs into 2 blocks that do not influence each other. We denote the job at the k-th position in the i-th block by $[i,k]$ and the number of jobs in block b by n_b. Solution of the single machine instance verifies whether a schedule exists for which, for a rational number U:

$$\sum_{b=1}^{2} \sum_{i=1}^{n_b} w_{[b,i]} \sum_{j=1}^{i-1} d_{[b,j]} \Big/ \sum_{i=1}^{n-1} d_i \le U.$$

We see that choice of the upper bound on the SAA-objective $U=L/\sum_j d_j$ will allow us to solve *D2S* by means of the single machine SAA instance. In conclusion, the single machine SAA stability problem is ordinarily *NP*-hard.

It is interesting to see that it is not the sequencing part itself but rather the interplay between sequencing and buffer allocation that induces the complexity status of this problem: if we select $\omega=\sum_j d_j$, then an adjacent interchange argument shows that the problem is solvable in polynomial time, even for multiple disruption

scenarios per activity, by ordering the jobs with shortest weighted expected disruption length (p_iEL_i/c_i) first.

4.2 Free number of parallel machines

Garey and Johnson [8] show the following problem to be *NP*-complete in the strong sense:

3-PARTITION. *Input: a finite set P of $3h$ elements ($h\in\mathbb{N}$), a bound $B\in\mathbb{N}$, and an integer size z_a for each $a\in P$, such that each z_a satisfies $B/4 < z_a < B/2$ and such that $\sum_{a\in P} z_a = hB$. Question: can P be partitioned into h disjoint sets $P_1,P_2,\ldots,P_h$ such that, for $1 \le i \le h$, $\sum_{a\in P_i} z_a = B$?*

$P||C_{max}$ is strongly *NP*-complete, since 3-PARTITION is a special case, so when the deadline can be restrictive, the complexity status of SAA with free number of parallel machines is immediate. Otherwise, strong *NP*-completeness for the sensitivity case is shown as follows. For an arbitrary instance of 3-PARTITION, we construct an instance of parallel machine SAA. We start from an arbitrary partition of P into 3-element subsets, and determine the size Ω of the largest subset. The jobs of the SAA are the elements of P, and $d_i=z_i$, $\forall i\in P$. We set $\omega = \Omega$ and give each job i a single disruption scenario $l_{i1}= \omega - B$. The bound U on the objective value is 0. All activities have equal probability of being disrupted. We have h parallel processors available.

Given our choice of ω, a feasible schedule for the corresponding instance of SAA always exists, so ω is not restrictive. Each activity should have a total float of $\omega - B$ in order to make $U=0$, and since no other cost coefficients are nonzero, we can always schedule all activities such that each processor is contiguously occupied from time 0. Time window $[B, \omega]$ should not be used by any activity, which leaves us with h separate time blocks on the h processors, each of length exactly B, and since this is just enough time in total to accommodate all the activities in P, each block must be completely filled; these blocks therefore play the same role as the sets $P_1, P_2,\ldots, P_h$ in the desired partition of P. We remind the reader that, since $B/4 < z_a < B/2$ for each item a, filling length B by means of 2 or 4 activities (or less or more, respectively), is impossible. Thus, the answer to the 3-PARTITION instance can be seen to be 'yes' if and only if the answer to SAA is 'yes' when bound U is set to 0. This description allows to specify a pseudo-polynomial transformation from 3-PARTITION.

For the stability case, a similar reduction can be set up, when additionally h enforcer jobs are included, indexed $k=3h+1,\ldots,4h$, with $c_k=2hl_{i1}+1$, $l_k=\omega+2hl_{i1}$ and $d_k=1$, and we set $c_i=1$, $i=1,\ldots,3h$, $U=2hl_{i1}$ and $\omega=\Omega+1$. If U is to be respected, each machine

should carry exactly one of the extra jobs, and this job should be scheduled last on the machine, more specifically from time ω to Ω. The remainder of the proof is similar to the sensitivity case, by noting that U can never by respected if any of the extra jobs is not protected completely from disruptions in its predecessors on the assigned machine.

4.3 Precedence constraints

Du et al. [7] show that problem $P2 \mid chains \mid C_{max}$ is strongly *NP*-hard, which allows for an easy reduction to the 2-machine SAA with precedence constraints, when the deadline can be restrictive. The non-restrictive sensitivity case is proven similarly, by addition of an extra job that is a successor to all jobs without successor. 1-machine sensitivity is polynomially solvable, similarly as in Section 4.1; remark that $1 \mid prec \mid C_{max}$ is also polynomially solvable (Lawler [13]).

The non-restrictive 1-machine SAA with precedence constraints and stability objective is *NP*-complete in the strong sense by reduction from the decision version of $1 \mid prec \mid \sum C_j$, which is strongly *NP*-hard (Lawler [14], Lenstra & Rinnooy Kan [15]). The reduction is simple: for an instance of the latter problem, we use the same jobs with durations equal to 1, ω equal to the number of jobs, all disruption probabilities equal, and single disruption lengths equal to the durations. All costs coefficients are also set equal, and the precedence constraints are maintained. The reduction is then immediate, by noting that no float can be inserted because of the choice of the deadline ω, and that the expected disruption length of a job is the sum of the original durations of the preceding jobs (=disruption lengths), times a constant (for normalizing the probabilities). The deadline is non-restrictive: any linear extension of the partial ordering implied by the precedence constraints defines a feasible schedule. This generalizes to the restrictive case.

4.4 Release dates

In this section we show the single machine SAA with non-restrictive deadline and release dates or 'ready times' to be strongly *NP*-complete for the stability objective. The restrictive sensitivity case is easily solved in polynomial time; an optimal schedule is obtained by starting the jobs in non-decreasing order of ready time, and as early as possible (again semi-active). This is straightforwardly shown by adjacent interchange argument.

Each instance of the strongly *NP*-complete decision version of $1 \mid r_i \mid \sum C_j$ (Lenstra et al. [16], referred to as $1rS$ in the following) can be solved by an instance of

the single machine SAA with release dates and stability objective. We start by the choice of a deadline ω for SAA: we select ω such that at least one optimal solution to $1rS$ exists with makespan no larger than ω, a safe value being the largest r_i-value plus the sum of all activity durations. The set of jobs to be scheduled consists of the activities from the $1rS$ instance with corresponding duration, collected in set P, augmented with $(\omega - \sum_{i \in P} d_i)$ enforcer activities of duration 1, which are gathered in set Q. Disruption lengths for all activities in $P \cup Q$ equal their duration; all disruption probabilities are equal. The cost coefficients c_i are zero for $i \in Q$ and 1 for $i \in P$; the ready times r_i are zero for $i \in Q$ and equal to the original ready times for $i \in P$.

For an arbitrary schedule to the resulting SAA instance, the expected increase in starting time for any activity j is proportional to its scheduled starting time, with proportionality constant $k=1/|P|$. The description easily allows to specify a pseudo-polynomial transformation from the decision version of $1rS$ to the single machine SAA with ready times, such that the stability version of the latter is seen to be strongly *NP*-complete: if L is the numerical bound on the objective function of a $1rS$-instance, SAA solves the instance with $U = k(L - \sum_{i \in P} d_i)$.

5. Conclusions

In this paper, we have analyzed the complexity of the generation of robust baseline schedules with the expected weighted deviation of activity starting times as stability measure, and with the objective of makespan protection. We assume that exactly one disturbance occurs when a schedule is implemented, in the form of an increase in the duration of a single activity. The underlying idea is that disturbances are sufficiently sparse and spread over time and throughout the project network. If abstraction is made of resource usage, it is has been shown earlier that the scheduling problem is solvable in polynomial time. This paper establishes complexity results for various machine scheduling sub-problems of the robust resource allocation problem. All complexity proofs have been formulated for the case with only one disruption scenario per activity.

References

[1] I. Adiri, J. Bruno, E. Frostig, A.H.G. Rinnooy Kan, Single machine flow-time scheduling with a single breakdown, Acta Informatica 36 (1989) 679-696.

[2] M.S. Akturk and E. Gorgulu, Match-up scheduling under a machine breakdown, European Journal of Operational Research 112 (1999) 81-97.

[3] J. Bruno, E.G.Jr. Coffmann, R. Sethi, Scheduling independent tasks to reduce mean finishing time, Communications of the ACM 17 (1974) 382-387.

[4] R.L. Daniels, J.E. Carrillo, β-robust scheduling for single-machine systems with uncertain processing times, IIE Transactions 29 (1997) 977-985.

[5] R.L. Daniels, P. Kouvelis, Robust scheduling to hedge against processing time uncertainty in single-stage production, Management Science 41 (1995) 363-376.

[6] A.J. Davenport, J.C. Beck, A survey of techniques for scheduling with uncertainty, unpublished manuscript available at http://www.eil.utoronto.ca/EIL/profiles/chris/zip/uncertainty-survey.ps.zip.

[7] J. Du, J.Y.-T. Leung, G.H. Young, Scheduling chain-structured tasks to minimize makespan and mean flow time, Information and Computation 92 (1991) 219-236.

[8] M.R. Garey, D.S. Johnson, Computers and intractability. A guide to the theory of NP-completeness, W.H. Freeman and company, 1979.

[9] J.N. Hagstrom, Computational complexity of PERT problems, Networks 18 (1988) 139-147.

[10] W. Herroelen and R. Leus, On the construction of stable project baseline schedules, Research Report 0220, Department of Applied Economics, Katholieke Universiteit Leuven, Belgium, 2002.

[11] W. Herroelen and R. Leus, Project scheduling under uncertainty – Survey and research potentials, Research Report 0225, Department of Applied Economics, Katholieke Universiteit Leuven, Belgium, 2002.

[12] G. Igelmund and F.J. Radermacher, Algorithmic approaches to preselective strategies for stochastic scheduling problems, Networks 13 (1983) 29-48.

[13] E.L. Lawler, Optimal sequencing of a single machine subject to precedence constraints, Management Science 19 (1973) 544-546.

[14] E.L. Lawler, Sequencing jobs to minimize total weighted completion time subject to precedence constraints, Annals of Discrete Mathematics 2 (1978) 75-90.

[15] J.K. Lenstra and A.H.G. Rinnooy Kan, Complexity of scheduling under precedence constraints, Operations Research 26 (1978) 22-35.

[16] J.K. Lenstra, A.H.G. Rinnooy Kan and P. Brucker, Complexity of machine scheduling problems, Annals of Discrete Mathematics 1 (1977) 343-362.

[17] R. Leus and W. Herroelen, Models for robust resource allocation in project scheduling, Research Report 0128, Department of Applied Economics, Katholieke Universiteit Leuven, Belgium, 2001.

[18] R. Möhring and F.J. Radermacher, The order-theoretic approach to scheduling: the stochastic case, Chapter 4 of Part III in: R. Slowinski and J. Weglarz, Advances in project scheduling, Elsevier, 1989.

[19] M. Pinedo, Scheduling. Theory, algorithms and systems, Prentice-Hall, 1995.

[20] F. Stork, Stochastic resource-constrained project scheduling, Ph.D. Thesis, Technische Universität Berlin, 2001.

[21] A.T. Unal, R. Uzsoy, A.S. Kiran, Rescheduling on a single machine with part-type dependent setup times and deadlines, Annals of Operations Research 70 (1997) 93-113.